(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 955 583 A2

# (12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
**10.11.1999 Bulletin 1999/45**

(51) Int Cl.[6]: **G06F 9/46**, G06F 9/30

(21) Application number: **99650039.3**

(22) Date of filing: **05.05.1999**

(84) Designated Contracting States:
**AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE**
Designated Extension States:
**AL LT LV MK RO SI**

(30) Priority: **06.05.1998 US 73500**

(71) Applicant: **SUN MICROSYSTEMS, INC.**
**Palo Alto, California 94303 (US)**

(72) Inventor: **Shaylor, Nicholas**
**Newark,CA 94560 (US)**

(74) Representative: **Hanna, Peter William Derek et al**
**Tomkins & Co.,**
**5 Dartmouth Road**
**Dublin 6 (IE)**

## (54) Context switching technique for processors with large register files

(57) A computer system and a method for operating a processor including the steps of establishing a first register save area (209a) and a second register save area (209b) in a memory (107), where each register save area holds data values that define a context, are described. The first context is loaded in the processor (102) by loading at least some of the data values from the first register save area into the plurality of registers (203). A first pointer value to the first register save area is stored in a current register file save area (RFSA) register (204). A context switch is indicated by storing a second pointer to the second register save area in the current RFSA register. The first pointer is transferred from the current RFSA register to a previous RFSA register. All of the data values that define the first context are transferred from the registers to a shadow register file (207). The second context is established in the processor by loading selected data values from the second register file save area into the plurality of registers.

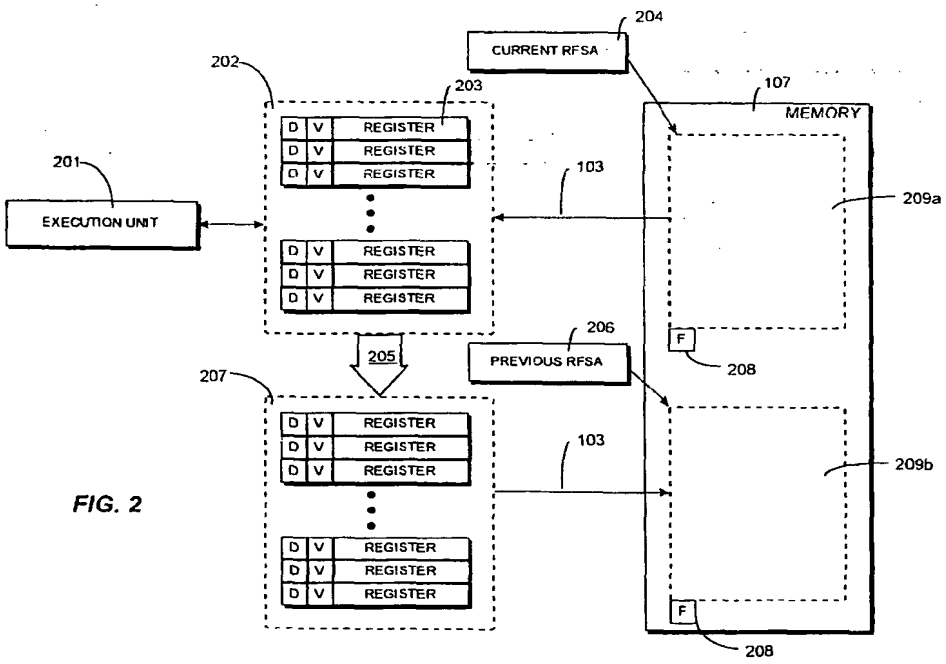


FIG. 2

EP 0 955 583 A2

## D scription

### 1. Field of th Invention.

[0001] The present invention relates, in general, to data processors, and, more particularly, to a method, apparatus and system for implementing context switching in processors with large register files.

### 2. Relevant Background.

[0002] Computer systems include central processing units (CPUs), microcontroller units (MCUs), and the like coupled with memory. Programs that run on such computer systems operate on data that may be stored and retrieved by the program or supplied at run-time. Programs include a plurality of saved instructions that define particular operations that are to be performed on the data.

[0003] Most processor architectures generally define a plurality of registers for holding the data to be operated on by the program instructions. These registers may be implemented as hardware registers or as register files in general purpose memory. The registers store both the instruction and data that are being or may be used by the processor. The registers are usually implemented in memory devices that are closely coupled with the processor to provide low-latency access to data required by the processor. The registers are typically defined in the processor architecture specification and so are usually considered part of the processor architecture even where they are physically implemented in another device.

[0004] High speed processors may have tens or hundreds of registers in a general register file. The large number of registers can enable the processor to process a large amount of data concurrently and to load or store data from longer latency storage before it is needed. Very long instruction word (VLIW) processors tend to have higher register number requirements because of the inherent parallelism of VLIW that results in more concurrent operations. The higher register number requirements place correspondingly higher bandwidth and response time requirements on the memory bus that transfers data between memory and the registers. It can take multiple memory bus clock cycles to transfer data into or remove data from all the available registers in a processor.

[0005] As a more specific example, multi-tasking and multi-threading processor architectures enhance data processing efficiency in many applications. In such architectures, software programs executing on the processor are segmented into atomic "threads" that execute on the processor. To ensure architectural integrity, each thread is normally guaranteed access to the entire register set defined by the processor architecture even if the thread only uses a fraction of that register set.

[0006] A particular thread's instructions and data, together with the architectural registers that store that data. are referred to as a "context". A "context switch" occurs when the architectural resources are switched from one thread to another thread. A context switch occurs, for example, when one thread become inactive or is terminated and the processor resources are applied to another active thread. A context switch also occurs, for example, when an executing thread accesses a resource that has a long latency, or when a thread with higher priority than the current thread is imposed on the processor. When a context switch occurs, the data in the registers is moved out of the registers and saved to persistent storage (or some other memory location). Data for the new context is then transferred into the registers.

[0007] One way to organize registers within a processor is to use a register windowing technique to access a plurality of registers in a register file. With register windowing, a register window has a predetermined number of contiguous registers, and the window can be moved linearly within the register file. At any one time, the register window permits program access to a subset of the total number of registers in the register file. Control registers are also associated with the register windows so that a program can manipulate the position of the window within the register file and monitor the status of the window.

[0008] For example, in the specification for a scaleable processor architecture, SPARC-V9, the general purpose registers for storing and manipulating data are arranged in register sets accessible through register windows, each register window having 32 registers. A particular processor can have multiple register sets ranging from three register sets to 32 register sets. Individual registers are addressable using a five-bit address in conjunction with a current window pointer (CWP). The register window is movable within the register sets such that a program can logically address multiple physical registers in the register sets by simply tracking a logical register name or specifier and the current window pointer.

[0009] In prior implementations, the entire register file is purged in response to a context switch, and the register file is initialized for a new process. If the new process is itself a saved process, the register values are restored from storage before the context takes effect. Because of this, two memory operations, one to write the old context to storage and a second to read the new context from storage, may be required for each context switch. For VLIW architectures, this situation creates an undesirable number of memory transactions that constitute overhead to the fundamental data processing performance of the processor. For example, in an architecture providing 256 registers, up to 512 memory transactions may be required to implement a context switch. This setup may be required in prior systems even where only a few of the 256 regist rs were actually used by the current process and where only a few of the registers will be used by the new process. A need exists for a



EPODOC ID: <EP 0955583A2 >

processor architecture that provides low overhead manipulation of a large register file.

**[0010]** Another limitation of existing processors is that during a context switch, all of the processors resources are dedicated to completing the context switch. Other operations are blocked until the new context is in place. This type of operation decreases the efficiency of the processor because every operation is stalled until all of the old context's registers are saved (including registers that were not used) and all of the new context's registers are initialized or restored (including registers that will not be used).

## SUMMARY OF THE INVENTION

**[0011]** Briefly stated, the present invention solves these and other limitations by saving only registers that have been modified in response to a context switch. Further, during a context switch, the new context's registers are dynamically loaded from its context record when the register is used. In this manner, no overhead penalty is incurred for registers that are architecturally specified, but not used by the thread. Also, the context saving process is performed in accordance with the present invention in parallel with other operations in the new context, minimizing the impact of context switching on processor performance.

**[0012]** In another aspect, the present invention involves a method for operating a processor including the steps of establishing a first register save area and a second register save area in a memory, where each register save area holds data values that define a context. The first context is loaded in the processor by loading at least some of the data values from the first register save area into the plurality of registers. A first pointer value to the first register save area is stored in a current RFSA register. A context switch is indicated by storing a second pointer to the second register save area in the current RFSA register. The first pointer is transferred from the current RFSA register to a previous RFSA register. All of the data values that define the first context are transferred from the registers to a shadow register file. The second context is established in the processor by loading selected data values from the second register file save area into the plurality of registers.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0013]**

FIG. 1 shows in block diagram form a computer system implementing context switching in accordance with the present invention; and

FIG. 2 shows in block diagram form components of a context switching apparatus in accordance with the present invention.

## DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

**[0014]** FIG. 1 illustrates in block diagram form a computer system incorporating an apparatus and system in accordance with the present invention. Processor architectures and computing systems are usefully represented as a collection of interacting functional units as shown in FIG. 1. These functional units, discussed in greater detail below, perform the functions of fetching instructions and data from memory, processing fetched instructions, managing memory transactions, interfacing with external I/O and displaying information.

**[0015]** The present invention is described in terms of an apparatus and method particularly useful in a computer system 100 such as shown in FIG. 1. FIG. 1 shows a typical general purpose computer system 100 incorporating a processor 102 and implementing both an application program and an operating system executing in processor 102. Computer system 100 in accordance with the present invention comprises a system bus 101 for communicating information, processor 102 coupled with bus 101 through input/output (I/O) devices within processor 102. Processor 102 is coupled to memory system 107 using a memory bus 103 to store information and instructions for processor 102. Memory system 107 comprises, for example, one or more levels of cache memory and main memory in memory unit 107. It should be understood that some cache memory is included on-chip with processor 102 in most applications in addition to cache and memory in memory system 107.

**[0016]** User I/O devices 106 are coupled to bus 101 and are operative to communicate information in appropriately structured form to and from the other parts of computer 100. User I/O devices may include a keyboard, mouse, magnetic or tape reader, optical disk, or other available I/O devices, including another computer. Mass storage device 117 is coupled to bus 101 and may be implemented using one or more magnetic hard disks, magnetic tapes, CD ROMs, large banks of random access memory, or the like. A wide variety of random access and read-only memory technologies are available and are equivalent for purposes of the present invention. Mass storage 117 includes computer programs and data stored therein. Some or all of mass storage 117 may be configured to be incorporated as part of memory system 107. Processor 102 includes a number of registers 203 (shown in FIG. 2) that are typically implemented in hardware. Data processing occurs by loading data into registers 203, modifying data in registers 203, and storing data from registers 203 back out to memory 107

**[0017]** A processor 102 with a large general register file 202 (e.g., 256 registers) carries with this the potential for 512 memory operations on a context switch. Many of these will be unnecessary and so may lengthen the context switch time considerately. The present invention seeks to reduce this time by only loading and saving registers that are in fact needed, and to perform the saving



BNSDOCID: <EP 0955583A2 1 >

operations in parallel with other instructions.

**[0018]** Referring to FIG. 2, an execution unit 201 within processor 102 access s (i.e., loads, stores, and modifies) data stored in general register file 202. General register file 202 comprises a plurality of registers 203 where the number of registers 203 is specified by the processor architecture. Each register 203 includes at least one, and typically more than one storage cell(s) each storing one bit of information. The registers 203 in general register file 202 are alternatively referred to as "architectural" register files. Each register 203 is accessible by execution unit 201 to enable processor 102 to load data from a selected register 203 and store data to a selected register 203.

**[0019]** The contents of registers 203 in aggregate define a "context" or the processor state during execution of a particular program thread or process. The context is saved persistently in memory 107 in a register save area 209a or 209b so that it can be restored on demand. Although only two register save areas 209a and 209b are shown, each context has a corresponding save area 209. Hence, there may be hundreds or thousands of register save areas 209.

**[0020]** As shown in FIG. 2, a current register file save area (RFSA) register 204 contains data pointing to an area of memory 107 where the current thread's register values are saved. The term "current thread" means a program thread currently executing in execution unit 201. In a particular example, RFSA 204 contains a pointer to a start location or first memory location of a sequentially block of memory holding all of the values from registers 203. Where the number of registers 203 in register file 202 is specified by the processor architecture, only the first register address is needed to access all of the registers 203 individually. Alternatively, the register values can be accessed from memory 107 in groups comprising some but less than all of the registers. A new value is written into current RFSA register 204 at the initiation of a context switch.

**[0021]** In accordance with the present invention, each of registers 203 include or is associated with a valid bit (labeled "V" in FIG. 2) indicating whether data of the register is effective data or not. When current RFSA register 204 is written to, indicating a context switch, all the general registers 203 are marked as invalid. If a register 203 is written to when in the invalid state the associated valid bit is then marked valid.

**[0022]** When a register 203 is read while in the invalid state, processor 102 automatically loads the value from the register file save area 209a of memory 107. When the value is loaded, the corresponding valid bit is marked valid. Each register 203 is also associated with a dirty bit (labeled "D" in FIG. 2) that indicates that the value stor d in the register file save ar a of memory 107 for that register 203 is out of sync with the current register 203. Registers having a dirty bit set are referred to herein as "dirty registers".

**[0023]** When current RFSA register 204 is written to (indicating a context switch) its current value is copied to previous RFSA regist r 206. Previous RFSA register 206 is, for example, coupled to current RFSA register 204 to receive the copied current RFSA value. Also, the value in all registers 203 are copied, preferably in one cycle, into shadow register file 207 over local register connection 205. In practice, shadow register file 207 may be implemented in a fashion so that each memory cell or storage location in each register 203 is physically adjacent to a corresponding memory cell or storage location in shadow register file 207. In this manner, memory bus 103 is not burdened with the traffic required to copy all of registers 203 during a context switch. The content of each register 203 is copied together with its associated valid bit and dirty bit. Once all the registers 203 have been copied to shadow register file 207, register file 202 can be dedicated to holding entries from the new context.

**[0024]** As there are unused cycles on the memory bus 103 registers values stored in shadow register 207 that are dirty as indicated by the dirty bit are written to the register save area 209b in memory 107 using memory bus 103 at a location described by or derived from the value held in previous RFSA 206. This "lazy" write back or commit of the register values in shadow register 207 makes efficient use of memory bus 103 and the available memory bus cycles. This function may be implemented without added processor instructions by hardware coupled to current RFSA register 204 to detect a change in current RFSA 204 value and coupled to shadow register 207 to detect whether any dirty bits are set in shadow register 207.

**[0025]** When a context switch occurs, two cases may exist. In a first case, if the value of the new context in current RFSA 204 is the same as the value in the previous RFSA 206, the register values in shadow register 207 and register file 202 can be swapped. In this case, it is not necessary to complete write back of all dirty registers in shadow register 207 to register save area 209b. To enable swapping of the register values in may be desirable to include a third register file to temporarily hold the contents during swapping, although other hardware mechanisms to implement this swapping will be apparent to those skilled in the design of memory and processor systems.

**[0026]** In the second case, the value of the new context identified by the value stored in current RFSA 204 is different from the value in previous RFSA register 206. In this case, all the registers having dirty bits currently set in shadow 207 (if any exist) must be saved to register save area 209b in memory 107 before the context switch takes place. In most applications, all of the dirty registers in shadow register 207 will have already been written back in free memory cycles. However, in applications where context switches occur without sufficient memory cycles to complete the write back, a forced write back is required. If the operating environment is such that frequent context switches occur with insufficient free mem-



BNSDOCID: <EP 0955583A2 I >

ory clock cycles to complete the write back of dirty registers in shadow 207, it may be desirable to steal memory cycles from the memory bus. The stolen memory cycles will impact normal operation, but will allow dirty registers in shadow register 207 to be completely written back before a subsequent context switch, resulting in a net improvement.

**[0027]** In most environments the write back of data from shadow register 207 to memory 107 will only write into a data cache. In uniprocessor environments it is acceptable for the contents of RFSA 209b to be held in cache because the processor can readily access cached data. However, in SMP environments, all contents of shadow register 207 having the dirty bits set must eventually be written to memory 107 so that the data is available to other processors in the SMP system. To ensure that all the registers 203 are eventually written out to memory 107, it may be necessary to force a cache write-back operation by issuing a commit instruction from time to time that writes all register values stored in cache to main memory. This commit instruction may be issued periodically or based upon elapsed time that the data has been in shadow register 207, or frequency of context switches, other equivalent criteria, or a combination of these criteria.

**[0028]** In a preferred implementation, the new register values are loaded over memory bus 103 as needed from the associated register save area 209a in memory 107 to register file 202 using the new value stored in current RFSA register 204. This is alternatively referred to as a "lazy" restore, a "dynamic" restore, or a "just-in-time" restore. This is in contrast to existing systems that load the values into all of registers 203 before beginning to process data stored in register file 202. As described hereinbefore, upon a context switch, the valid bits of each register 203 are marked invalid. As execution unit 201 requests data for the first time from a register 203, the value is automatically loaded from memory 107 (because the register 203 is marked invalid). Some delay can be added to the instruction execution until the restored data is ready. Until the data from a particular register is requested, it remains in main memory (or cache). This dynamic restore feature of the present invention saves memory bus cycles that are ordinarily used to transfer register values that are not used by the executing thread.

**[0029]** Because register entries are dynamically loaded on use, the registers that are loaded are available for use very quickly, without the latency normally associated with restoring the entire register file 202. Hence, the dynamic loading feature enables execution unit 201 to process data from registers 203 faster than prior processors. By avoiding the latency required by restoring the entire register file 202, the processor 102 in accordance with the present invention operates with greater efficiency.

**[0030]** The method in accordance with the present invention carries an added advantage in a symmetric multiprocessing (SMP) environment. In an SMP environment, a thread from a single threaded program may execute on different processors, each having distinct register file 202. In these cases a CPU may attempt to execute a task that has not finished having its registers written back to memory on another CPU.

**[0031]** In the prior art, this problem could be handled in three ways: 1) save the registers 203 at every context switch and just perform the lazy restore; 2) bind the switched out thread to the processor upon which it was executed and which holds its shadow register 207; and 3) interrupt the processor having the shadow register 207 and force that processor to write the shadow register 207 to it associated register save area 209b in memory. All of these have disadvantages, but in accordance with a preferred embodiment of the present invention, shadow register file 207 will automatically synchronize with its associates register save area 209b given sufficient time (i.e., sufficient available memory cycles). Each register file save area desirably includes a flag 208 comprising a single bit to show that synchronization is complete (i.e., all registers in shadow register file 207 having a dirty bit have been written back to memory 107. This bit flag 208 can be tested by a processor 102 that request to load the context stored in shadow register 207.

**[0032]** Although the invention has been described and illustrated with a certain degree of particularity, it is understood that the present disclosure has been made only by way of example, and that numerous changes in the combination and arrangement of parts can be resorted to by those skilled in the art without departing from the spirit and scope of the invention, as hereinafter claimed.

## Claims

1. A computer system including a processor and a memory, the computer system comprising:

   a general register file including a plurality of general registers holding data;
   an execution unit within the processor operable to access data from selected general registers;
   a first register save area located at an address in the memory, the first register save area storing data held in the plurality of registers;
   a current register file save area register holding a value pointing to the location of the first register save area in memory;
   a shadow register file coupled to receive data from the general register file and to send data to the memory, the shadow register file having an entry for each of the general registers;
   a second register save area located at an address in the memory, the second register save area storing data held in the shadow register



EPODOC: <EP 0955583A2 1 >

file; and
a previous register file save area register coupled to the current register file save area register and holding a value pointing to the location of the second register save area in memory.

2. The computer system of claim 1 wherein the general register file holds data for one of a plurality of contexts.

3. The computer system of claim 1 wherein each general register is associated with a valid bit indicating whether data of the associated general register is effective data.

4. The computer system of claim 1 wherein each general register is associated with a dirty bit indicating whether data of the associated general register is synchronized with data corresponding to the associated general register in the register save area

5. The computer system of claim 4 further comprising
a flag associated with each general register save area, wherein the flag indicates when all values in the register save area are synchronized

6. The computer system of claim 1 further comprising a memory bus coupling the plurality of general registers to the memory.

7. The computer system of claim 6 further comprising a register bus separate from the memory bus coupling the general register file to the shadow register file.

8. The computer system of claim 1 further comprising a register bus coupling the general register file to the shadow register file wherein each general register comprises a plurality of one-bit storage locations and the register bus comprises a connection from each storage location in the general register file to a corresponding storage location in the shadow register file to enable the data held in all of the plurality of registers to be transferred to the shadow register file in a single clock cycle.

9. The computer system of claim 1 further comprising:
a memory bus coupling the plurality of general registers to the memory, wherein the shadow register file sends data to the memory using the memory bus while the memory bus is not being used to communicate data between the plurality of registers and the memory.

10. The computer system of claim 1 further comprising:
a control connection between the shadow register file, the current RFSA register and the previous RFSA register, wherein the shadow register file is further coupled to send data to the plurality of registers in response to detecting that the current RFSA value equals the previous RFSA value.

11. A method for operating a processor having a plurality of registers holding data and a memory, the method comprising the steps of:

providing a first register save area in the memory, the first register save area holding data values that define a first context;
providing a second register save area in the memory, the second register save area holding data values that define a second context;
providing the first context in the processor by loading at least some of the data values from the first register save area into the plurality of registers;
storing a first pointer value which points to the first register save area in a current RFSA register;
indicating a context switch by storing a second pointer to the second register save area in the current RFSA register;
transferring the first pointer from the current RFSA register to a previous RFSA register;
transferring substantially all of the data values that define the first context from the registers to a shadow register file; and
providing the second context in the processor by loading selected data values from the second register file save area into the plurality of registers.

12. The method of claim 11 further comprising the step of:
providing a plurality of register file save areas in the memory wherein each register file save area holds data values that define a unique context, and

13. The method of claim 12 wherein the first register file save area and the second register file save area are selected from the plurality of register file save areas by a thread executing on the processor.

14. The method of claim 11 wherein the step of transferring substantially all of the data values further comprises the step of:
copying each stored bit in the data values that define the first context to a corresponding bit storage location in the shadow register in a single memory cycle.

15. The method of claim 11 further comprising the step of:
marking each of the plurality of registers with a valid bit indicating whether the data of the register is effective data.



BNSDOCID: <EP 0955583A2 I >

16. The method of claim 11 further comprising the step of:

marking each of the plurality of registers with a dirty bit indicating whether the data of the register is synchronized with data in its corresponding register file save area.

17. The method of claim 11 further comprising the step of:

marking each register save area with a flag indicating whether substantially all values in the register file save area are synchronized.

18. The method of claim 11 wherein the step of providing the second context further comprises loading the data values from the second register file save area on a register by register basis as needed by a thread executing on the processor.

19. The method of claim 11 wherein the step of providing the second context further comprises the steps of:

marking each of the plurality of registers with a valid bit indicating whether the data of the register is effective data:
setting each of the valid bits to indicate that the data is not effective data;
each time a register is accessed, retrieving data from the accessed register if the valid bit indicates that the data is effective; and
each time a register is accessed, retrieving data from the register file save area if the valid bit indicates that the data is not effective, filling the register with the retrieved data, and setting the valid bit for the register to indicate that the register's data is now effective.

20. A computer program product for use with a general purpose computer, said computer program product comprising:

a computer usable medium having computer readable program code means embodied in said medium for operating a processor having a plurality of registers holding data and a memory, the computer program product having:
computer readable program code devices for causing a computer to provide a first register save area in the memory, the first register save area holding data values that define a first context;
computer readable program code devices for causing a computer to provide a second register save area in the memory, the second register save area holding data values that define a second context;
computer readable program code devices for causing a computer to provide the first context in the processor by loading at least some of the data values from the first register save area into the plurality of registers;
computer readable program code devices for causing a computer to store a first pointer value to the first register save area in a current RFSA register;
computer readable program code devices for causing a computer to indicate a context switch by storing a second pointer to the second register save area in the current RFSA register;
computer readable program code devices for causing a computer to transfer the first pointer from the current RFSA register to a previous RFSA register;
computer readable program code devices for causing a computer to transfer all of the data values that define the first context from the registers to a shadow register file; and
computer readable program code devices for causing a computer to provide the second context in the processor by loading selected data values from the second register file save area into the plurality of registers.

21. A register architecture for a data processor comprising:

a memory having a plurality of addressable storage locations;
a general register file including a plurality of general registers; and
a plurality of register save areas defined in the memory, each register save file holding data defining a unique context and each register addressable to provide data to the general register file on a register-by-register basis.

22. The register architecture of claim 21 further comprising:

a dirty bit associated with each general register indicating whether data in the register is synchronized with data in the register save area; and
a register write back device data to write back only registers marked as dirty to the register save file during a context switch.

23. A method for operating a processor having a plurality of registers holding data defining a current context, wherein the processor is associated with a memory, the method comprising the steps of:

providing a plurality of register save files located in the memory, each register save file holding data defining a particular context;



BNSDOCID: <EP_. 0955583A2 I >

indicating a new context by pointing to a location of a selected register save file; and
loading the new context on a register-by-register basis as each register is required by a thread operating on the processor, wherein at least one but less than all of the registers are loaded.

**24.** A computer data signal embodied in a carrier wave comprising:

a first code portion comprising code configured to cause a computer to define a plurality of register save files located in the computer memory, each register save file holding data defining a particular context;
a second code portion comprising code configured to cause a computer to indicate a new context by pointing to a location of a selected register save file; and
a third code portion comprising code configured to cause a computer to load the new context on a register-by-register basis as each register is required by a thread operating on the computer such that at least one but less than all of the registers are loaded.



BNSDOCID: <EP___0955583A2_I_>

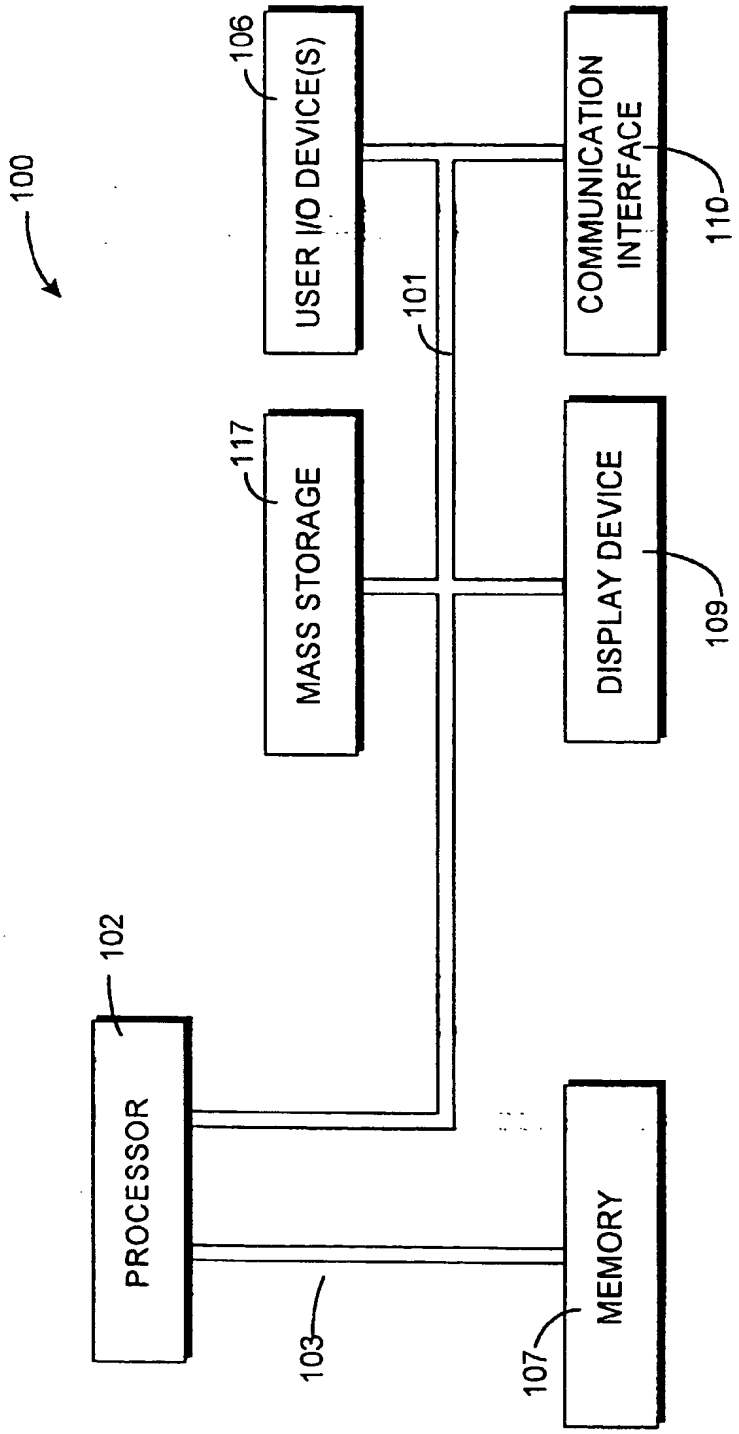


FIG. 1



BNSDOCID: <EP___0955583A2_I_>

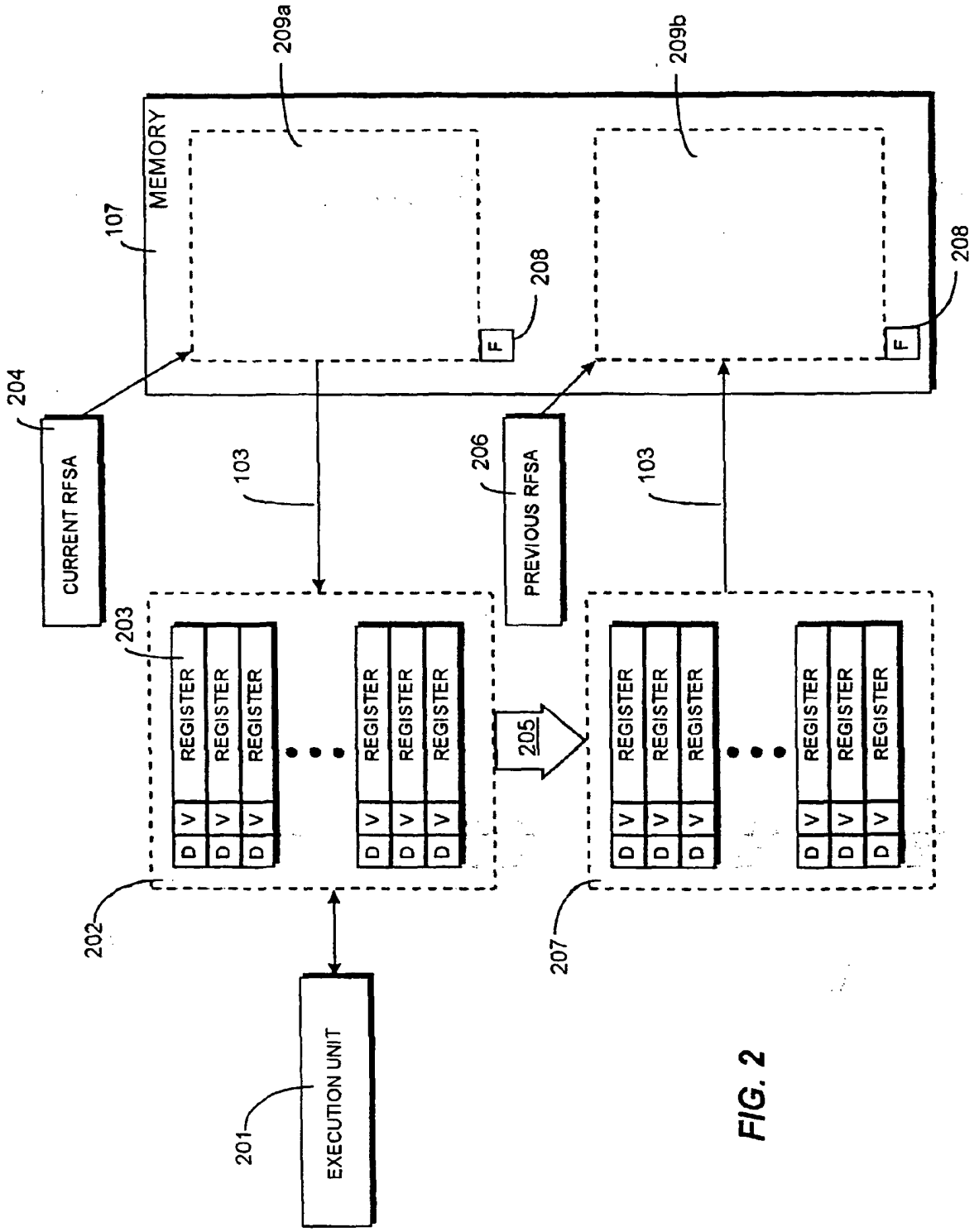


FIG. 2



BNSDOCID: <EP___0955583A2_I_>

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 955 583 A3

# (12) EUROPEAN PATENT APPLICATION

(88) Date of publication A3:
**11.04.2001 Bulletin 2001/15**

(43) Date of publication A2:
**10.11.1999 Bulletin 1999/45**

(21) Application number: **99650039.3**

(22) Date of filing: **05.05.1999**

(51) Int Cl.[7]: **G06F 9/46**, G06F 9/30

(84) Designated Contracting States:
**AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE**
Designated Extension States:
**AL LT LV MK RO SI**

(30) Priority: **06.05.1998 US 73500**

(71) Applicant: **SUN MICROSYSTEMS, INC.**
**Palo Alto, California 94043 (US)**

(72) Inventor: **Shaylor, Nicholas**
**Newark,CA 94560 (US)**

(74) Representative:
**Parkes, Andrew John Aykroyd et al**
**c/o Tomkins & Co.**
**5 Dartmouth Road**
**Dublin 6 (IE)**

## (54) Context switching technique for processors with large register files

(57) A computer system and a method for operating a processor including the steps of establishing a first register save area (209a) and a second register save area (209b) in a memory (107), where each register save area holds data values that define a context, are described. The first context is loaded in the processor (102) by loading at least some of the data values from the first register save area into the plurality of registers (203). A first pointer value to the first register save area is stored in a current register file save area (RFSA) register (204). A context switch is indicated by storing a second pointer to the second register save area in the current RFSA register. The first pointer is transferred from the current RFSA register to a previous RFSA register. All of the data values that define the first context are transferred from the registers to a shadow register file (207). The second context is established in the processor by loading selected data values from the second register file save area into the plurality of registers.

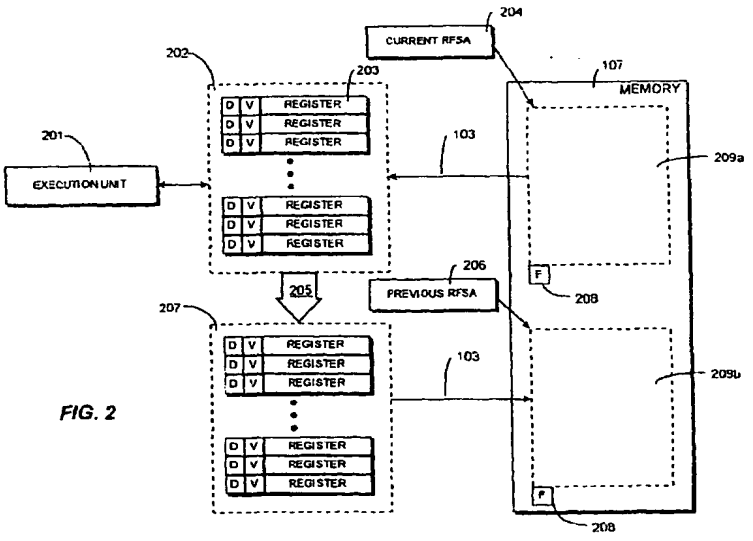


FIG. 2

EP 0 955 583 A3



EPODOC <EP 0955583A3 1 >

European Patent Office

# EUROPEAN SEARCH REPORT

Application Number
EP 99 65 0039

## DOCUMENTS CONSIDERED TO BE RELEVANT

| Category | Citation of document with indication, where appropriate, of relevant passages | Relevant to claim | CLASSIFICATION OF THE APPLICATION (Int.Cl.6) |
|---|---|---|---|
| Y | EP 0 373 790 A (TOKYO SHIBAURA ELECTRIC CO) 20 June 1990 (1990-06-20)<br>* abstract *<br>* column 2, line 4 - line 38; figure 1 *<br>* column 3, line 51 - column 4, line 18; figure 2 *<br>* column 5, line 27 - column 6, line 30; figures 6,7 * | 1,2,6-8, 11-14, 20,21, 23,24 | G06F9/46<br>G06F9/30 |
| Y | US 5 349 680 A (FUKUOKA KATSUHITO) 20 September 1994 (1994-09-20)<br>* abstract; figure 1 *<br>* column 4, line 1 - column 6, line 34 *<br>* claims 1,2,5,6 * | 1,2,6-8, 11-14, 20,21, 23,24 | |
| A | "Technique for Reducing the Number of Registers Saved at a Context Swap" IBM TECHNICAL DISCLOSURE BULLETIN,US,IBM CORP. NEW YORK, vol. 33, no. 3A, August 1990 (1990-08), pages 234-235, XP000123918 ISSN: 0018-8689<br>* the whole document * | 1-24 | TECHNICAL FIELDS SEARCHED (Int.Cl.6)<br>G06F |
| A | EP 0 285 310 A (TOKYO SHIBAURA ELECTRIC CO) 5 October 1988 (1988-10-05)<br>* abstract *<br>* page 3, line 47 - page 4, line 13 * | 1,4,11, 16,20-23 | |

-/—

The present search report has been drawn up for all claims

| Place of search | Date of completion of the search | Examiner |
|---|---|---|
| THE HAGUE | 6 February 2001 | Wiltink, J |

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)



BNSDOCID: <EP 0955583A3 I >

European Patent Office

# EUROPEAN SEARCH REPORT

Application Number
EP 99 65 0039

## DOCUMENTS CONSIDERED TO BE RELEVANT

| Category | Citation of document with indication, where appropriate, of relevant passages | Relevant to claim | CLASSIFICATION OF THE APPLICATION (Int.Cl.6) |
|---|---|---|---|
| A | DESAI S M: "General Purpose Register Extension" IBM TECHNICAL DISCLOSURE BULLETIN, vol. 24, no. 3, August 1981 (1981-08), pages 1404-1405, XP002159532 IBM CORP., NEW YORK, US ISSN: 0018-8689 * the whole document * | 1,11,20, 21,23 | |
| A | US 4 594 660 A (GUENTHNER ET AL.) 10 June 1986 (1986-06-10) * abstract * * column 2, line 15 - column 3, line 11 * * column 4, line 64 - column 5, line 16 * * column 9, line 26 - line 41 * | 1,6-9, 11,14,20 | |
| | | | TECHNICAL FIELDS SEARCHED (Int.Cl.6) |

The present search report has been drawn up for all claims

| Place of search | Date of completion of the search | Examiner |
|---|---|---|
| THE HAGUE | 6 February 2001 | Wiltink, J |

CATEGORY OF CITED DOCUMENTS

X: particularly relevant if taken alone
Y: particularly relevant if combined with another document of the same category
A: technological background
O: non-written disclosure
P: intermediate document

T: theory or principle underlying the invention
E: earlier patent document, but published on, or after the filing date
D: document cited in the application
L: document cited for other reasons

&: member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)



BNSDOCID: <EP 0955583A3 >

## ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO. EP 99 65 0039

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

06-02-2001

| Patent document cited in search report | Publication date | Patent family member(s) | Publication date |
|---|---|---|---|
| EP 0373790 A | 20-06-1990 | JP 2148223 A | 07-06-1990 |
| US 5349680 A | 20-09-1994 | JP 4172551 A | 19-06-1992 |
| | | JP 2062324 C | 24-06-1996 |
| | | JP 4182834 A | 30-06-1992 |
| | | JP 7095276 B | 11-10-1995 |
| | | KR 9512293 B | 16-10-1995 |
| EP 0285310 A | 05-10-1988 | JP 63245529 A | 12-10-1988 |
| | | DE 3889578 D | 23-06-1994 |
| | | DE 3889578 T | 01-09-1994 |
| | | KR 9201101 B | 01-02-1992 |
| | | US 5021993 A | 04-06-1991 |
| US 4594660 A | 10-06-1986 | AU 571461 B | 21-04-1988 |
| | | AU 2007983 A | 19-04-1984 |
| | | CA 1205565 A | 03-06-1986 |
| | | DE 3377027 D | 14-07-1988 |
| | | EP 0106670 A | 25-04-1984 |
| | | JP 1508409 C | 26-07-1989 |
| | | JP 59091547 A | 26-05-1984 |
| | | JP 63053571 B | 24-10-1988 |

DOCKET NO: P20010025
SERIAL NO:
APPLICANT: Christean May et al.
LERNER AND GREENBERG P.A.
P.O. BOX 2480
HOLLYWOOD, FLORIDA 33022
TEL. (954) 925-1100

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82



EPODOC: EP 0955583A3 1 >